Implémenter For Each

Le principe

Lorsque l'on fait un *For Each* sur une classe, on demande d'énumérer des *Variant* ou des objets. VB demande à la fonction *NewEnum*, qui a l'ID (dans outils/attributs de procédure) –4, de renvoyer un objet *IEnumVARIANT*. VB appelle la méthode *Next* de cet objet pour demander le prochain élément jusqu'à ce qu'elle renvoie 1.

Si l'on énumère une collection interne

Si l'on a une collection qui est déclarée comme suit :

```
Private m_Items As Collection
```

On créera une fonction publique NewEnum :

```
Public Function NewEnum() As IEnumVARIANT
    Set NewEnum = m_Items.[_NewEnum]
End Function
```

Il faut modifier les attributs de la fonctions : Menu *Outils|Attributs de procédure…*

- Cliquer sur *Avancés >>*
- Dans *ID de la procédure* : taper –4
- Cocher la case *Masquer ce membre* afin de la réserver à VB uniquement

Voilà, c'est tout…mais imaginons que l'objet de l'énumération ne soit pas une collection VB…

Si l'on énumère autre chose

Le problème est que cette interface expose des types que VB ne supporte pas dans du code. On ne peut donc pas faire un simple *Implements.*
Il existe deux solutions :

- Créer une typelib VB-izée
- Utiliser un objet léger sans typelib puisque que VB définit et autorise une variable ou fonction de type *IEnumVARIANT* mais pas son utilisation dans du code.

Nous allons nous servir des objets légers afin de créer un objet *IEnumVARIANT* pour gérer *Next* par nous même. On utilisera les objets alloués dans le tas.

Le principe est le suivant :

- NewEnum renvoie une instance de notre objet léger à VB
- VB appelle uniquement la méthode Next tant qu'elle renvoie 0. Quand il n'y a plus d'éléments à énumérer, *Next* renvoie 1.
- A chaque appel de *Next*, cette méthode rappelle la méthode *ForEach* de l'objet qui expose la collection. Celle-ci remplit le *Variant* avec l'élément demandé et renvoie 0 (élément existant) ou 1 (élément inexistant).

Le code sera le suivant :

```
'la structure d'un Guid
Private Type Guid
```

```
    Data1 As Long
    Data2 As Integer
    Data3 As Integer
    Data4(0 To 7) As Byte
End Type

'VB se sert de l'interface IEnumVARIANT pour implémenter For Each
'Vous pouvez énumérer tout type de données sauf les Type que vous définissez :
' vous ne pouvez pas affecter une structure à un Variant (sauf dans les dll ActiveX)

'implémentation d'une interface IEnumVARIANT pour For Each

'indique que l'objet ne supporte pas l'interface demandé
Private Const E_NOINTERFACE As Long = &H80004002
'indique que la méthode n'est pas implémentée
Private Const E_NOTIMPL As Long = &H80004001

'le guid de l'interface IEnumVARIANT
Private Const guidIEnumVARIANT As String = "{00020404-0000-0000-C000-
000000000046}"
'le guid numérique
Private m_guidIEnumVARIANT As Guid

'la vtable : table des fonctions virtuelles
'en résumé, elle contient les pointeurs ves les fonctions publiques de l'objet
'dans notre cas, il faut 7 fonctions :
'   -> les 3 fonctions de l'interface de base IUnknown (base du fonctionnement de COM)
'   -> les 4 fonctions de l'interface IEnumVARIANT
Private Type VTable
    Methods(0 To 6) As Long
End Type

'on garde une trace du pointeur vers la vtable pour savoir si elle est initialisée
Private m_pVTable As Long
'on garde une variable contenant la vtable
Private m_VTable As VTable

'ceci est la structure d'un objet : la seule différence entre un objet et un Type,
'c'est le premier membre, un pointeur vers la vtable
Private Type EnumVar
    'le pointeur vers la vtable
    pVTable As Long
    'le compteur de référence pour savoir quand on doit libérer la mémoire de l'objet
    cCount As Long

    'données attachées
    '-------------------
    'l'objet de rappel pour la fonction Next
    lpCollection As Object
    'l'index courant de l'énumération
    iCurrent As Long
End Type
```

```
'remplit une zone avec des 0
Private Declare Sub ZeroMemory Lib "kernel32.dll" Alias "RtlZeroMemory" (ByRef
Destination As Any, ByVal Length As Long)
'copie d'une zone mémoire dans une autre
Private Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (ByRef
Destination As Any, ByRef Source As Any, ByVal Length As Long)
'alloue de la mémoire dynamiquement dans le tas
Private Declare Function CoTaskMemAlloc Lib "ole32.dll" (ByVal cb As Long) As Long
'libère de la mémoire allouée dans le tas
Private Declare Sub CoTaskMemFree Lib "ole32.dll" (ByRef pv As Any)

'convertit un guid chaine en guid numérique
Private Declare Sub CLSIDFromString Lib "ole32.dll" (ByVal lpsz As Long, ByRef pclsid
As Guid)

'compare deux guids et renvoie True s'ils sont égaux
Private Function IsIIDEqual(guid1 As Guid, guid2 As Guid) As Boolean
    IsIIDEqual = ((guid1.Data1 = guid2.Data1) And _
                  (guid1.Data2 = guid2.Data2) And _
                  (guid1.Data3 = guid2.Data3) And _
                  (guid1.Data4(0) = guid2.Data4(0)) And _
                  (guid1.Data4(1) = guid2.Data4(1)) And _
                  (guid1.Data4(2) = guid2.Data4(2)) And _
                  (guid1.Data4(3) = guid2.Data4(3)) And _
                  (guid1.Data4(4) = guid2.Data4(4)) And _
                  (guid1.Data4(5) = guid2.Data4(5)) And _
                  (guid1.Data4(6) = guid2.Data4(6)) And _
                  (guid1.Data4(7) = guid2.Data4(7)))
End Function

'comme on ne peut pas affecter la valeur de AddressOf à une variable directement,
'on doit le passer en paramètre d'une fonction qui nous le renvoit
Public Function FuncPtr(ByVal addr As Long) As Long
    FuncPtr = addr
End Function

'crée un objet IEnumVARIANT qui appelera la méthode ForEach de l'objet objCallback à
chaque itération de For Each
Public Function InitCollection(ByVal objCallback As Object) As IEnumVARIANT
    'pointeur vers la collection
    Dim ptrCollection As Long
    'contenu de l'objet
    Dim Collection As EnumVar

    'si la vtable n'est pas initialisée
    If m_pVTable = 0 Then
        'on la remplie
        With m_VTable
            .Methods(0) = FuncPtr(AddressOf QueryInterface)
            .Methods(1) = FuncPtr(AddressOf AddRef)
            .Methods(2) = FuncPtr(AddressOf Release)
```

```
            .Methods(3) = FuncPtr(AddressOf IEnumVARIANT_Next)
            .Methods(4) = FuncPtr(AddressOf IEnumVARIANT_Skip)
            .Methods(5) = FuncPtr(AddressOf IEnumVARIANT_Reset)
            .Methods(6) = FuncPtr(AddressOf IEnumVARIANT_Clone)
        End With
        'et on en garde l'adresse
        m_pVTable = VarPtr(m_VTable)

        'on initialise le guid numérique
        CLSIDFromString ByVal StrPtr(guidIEnumVARIANT), m_guidIEnumVARIANT
    End If

    'on contruit l'objet
    With Collection
        'le pointeur vers la vtable
        .pVTable = m_pVTable
        'le compteur de référence : on crée un objet donc il est à un
        .cCount = 1

        'on garde une trace de l'objet de rappel
        Set .lpCollection = objCallback
    End With

    'on alloue de l'espace mémoire pour l'objet
    ptrCollection = CoTaskMemAlloc(LenB(Collection))
    'si succès
    If ptrCollection Then
        'on remplit l'objet
        CopyMemory ByVal ptrCollection, ByVal VarPtr(Collection), LenB(Collection)
    End If

    'on assigne la référence à la variable de retour de la fonction
    CopyMemory ByVal VarPtr(InitCollection), ptrCollection, 4&

    'on évite la libération des ressources que nous avons transférées dans l'objet
    ZeroMemory ByVal VarPtr(Collection), LenB(Collection)
End Function

'implémentation des méthodes de l'interface
'le premier paramètre de toutes les méthodes est un pointeur vers la zone mémoire
'   servant à stocker le pointeur de vtable et les données (l'instance de l'objet)

'cette fonction sert à demander à l'objet s'il sait gérer l'interface iid (c'est un GUID)
'si oui, elle retourne l'adresse vers l'objet du type en question dans ppvObject et S_OK
'si non, elle met Nothing (cad 0) dans ppvObject et renvoie E_NOINTERFACE
Private Function QueryInterface( _
  ByRef This As EnumVar, _
  ByRef iid As Guid, _
  ByRef ppvObject As Long _
) As Long
    If IsIIDEqual(m_guidIEnumVARIANT, iid) Then
        This.cCount = This.cCount + 1
```

```
        ppvObject = VarPtr(This)
        QueryInterface = 0
    Else
        ppvObject = 0
        QueryInterface = E_NOINTERFACE
    End If
End Function

'cette fonction incrémente un compteur de référence (nombre d'instance) de l'objet
Private Function AddRef(ByRef This As EnumVar) As Long
This.cCount = This.cCount + 1
AddRef = This.cCount
End Function

'cette fonction décrémente un compteur de référence (nombre d'instance) de l'objet
'quand le compteur atteind 0, sa structure est libérée
Private Function Release(ByRef This As EnumVar) As Long
This.cCount = This.cCount - 1
Release = This.cCount
If This.cCount = 0 Then
    Set This.lpCollection = Nothing
    CoTaskMemFree ByVal VarPtr(This)
End If
End Function

'implémentation de la méthode d'énumération

Private Function IEnumVARIANT_Next( _
  ByRef This As EnumVar, _
  ByVal celt As Long, _
  ByRef rgVar As Variant, _
  ByVal pCeltFetched As Long _
) As Long
    Dim lng As Long

    'normalement, VB met toujours celt à 1 :
    'il réclame toujours un et un seul élément de la collection
    If celt <> 1 Then
        IEnumVARIANT_Next = E_NOTIMPL
        Exit Function
    End If

    'si l'appelant demande le nombre d'éléments renvoyés
    If pCeltFetched Then
        'on en renvoie toujours 1
        lng = 1
        CopyMemory ByVal pCeltFetched, lng, 4&
    End If

    'on appelle la fonction de l'objet possédant la collection pour qu'elle renvoie l'élément
courant
    'Elle appelle la fonction publique ForEach de l'objet
```

```
    '-----------------------------------------------------------------------------
    'ForEach doit être une fonction PUBLIQUE définie dans l'objet qui contient la collection
    'Public Function ForEach(ByVal iCurrent as Long,ByRef var as Variant) as Long
    IEnumVARIANT_Next = This.lpCollection.ForEach(This.iCurrent, rgVar)

    'on passe à l'élément suivant
    This.iCurrent = This.iCurrent + 1
End Function

'fait une copie de l'objet d'énumération
'cette fonction n'est jamais appelée par VB
Private Function IEnumVARIANT_Clone( _
  ByRef This As EnumVar, _
  ByRef ppEnum As Long _
) As Long

    ppEnum = CoTaskMemAlloc(LenB(This))
    CopyMemory ByVal ppEnum, This, LenB(This)

    IEnumVARIANT_Clone = 0
End Function

'cette fonction remet à 0 l'énumération
Private Function IEnumVARIANT_Reset(ByRef This As EnumVar)
    This.iCurrent = 0
End Function

'cette fonction avance de celt éléments sans les lire
Private Function IEnumVARIANT_Skip( _
  ByRef This As EnumVar, _
  ByVal celt As Long _
) As Long
    This.iCurrent = This.iCurrent + celt
End Function
```

Notons les points suivants :

- La gestion de la mémoire de l'objet se fait dans Release, la construction étant celle d'un objet dans le tas
- *QueryInterface* accepte seulement l'interface *IEnumVARIANT*
- L'interface *IEnumVARIANT* a quatre méthodes :
    - *Next*, c'est la seule que VB utilise. Elle est appelée pour renvoyer les éléments un par un dans la boucle *For Each* tant qu'elle renvoie 0
    - *Clone*, elle n'est jamais appelée par VB. Elle permet de dupliquer l'objet *IEnumVARIANT* afin de sauvegarder son état
    - *Reset*, elle n'est jamais appelée par VB. Elle permet de redémarrer l'énumération des éléments depuis le début
    - *Skip*, elle n'est jamais appelée par VB. Elle permet de passer *n* éléments sans les lire.
- Pour les méthodes jamais appelées, on pourrait se contenter de renvoyer *E_NOIMPL* pour dire que l'on ne les gère pas.

Utilisation dans un module de classe

Dans le module de classe le code sera le suivant :

```
'cette méthode est appelée par VB pour demander un objet d'énumération
'elle doit avoir un ID de -4 dans les attributs de procédure
Public Function NewEnum() As IEnumVARIANT
    'on renvoie l'objet d'énumération
    Set NewEnum = InitCollection(Me)
End Function

'renvoie l'élément d'index iCurrent dans la variable var
'elle doit renvoyer 0 si tout va bien
'                   1 s'il n'y a plus d'éléments dans la collection
Public Function ForEach(ByVal iCurrent As Long, var As Variant) As Long
'
End Function
```

Notons les points suivants :

- Il faut modifier les attributs de la fonctions : Menu *Outils|Attributs de procédure…*
    - Cliquer sur *Avancés >>*
    - Dans *ID de la procédure* : taper –4
    - Cocher la case *Masquer ce membre* afin de la réserver à VB uniquement
- La fonction *ForEach* doit être publique afin d'être appelable depuis l'extérieur : l'objet *IEnumVARIANT*
- La fonction *ForEach* doit absolument être implémentée dans l'objet exposant la collection par *NewEnum*.
    - *iCurrent* est l'index (partant de 0) de l'élément à renvoyer dans *var*
    - *var* doit contenir l'élément *iCurrent* (s'il existe)
    - La fonction renvoie :
        - 1 s'il n'y a plus d'éléments à énumérer (si *iCurrent* n'existe pas)
        - 0 s'il y a encore des éléments à énumérer (si *iCurrent* existe)